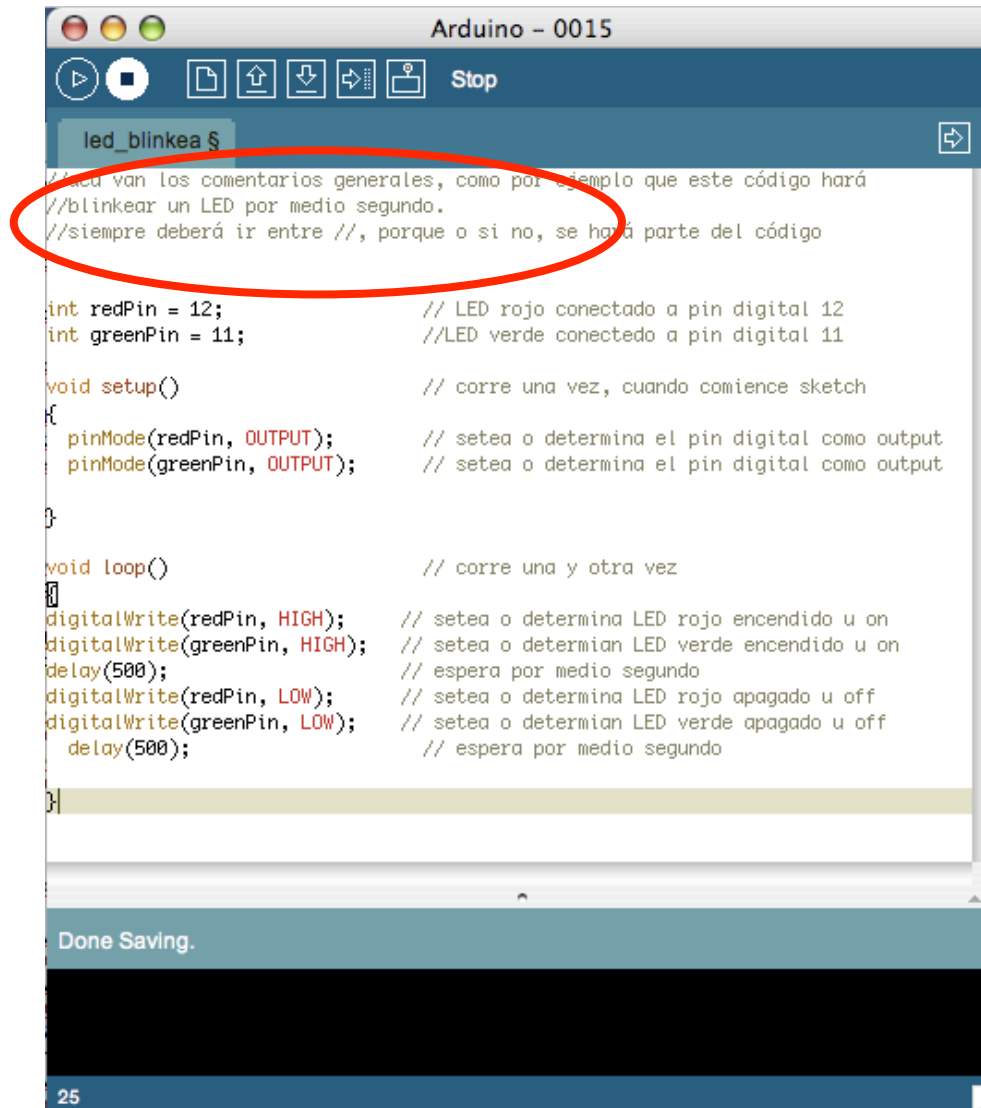


Conceptos Básicos de Programación

```
Put a pin into input mode
Loop
Check if the pin is 0 or 5 volts
If the pin has 5 volts
Do something
End if
End Loop
```

2.1. Estructura de un programa para Arduino



```
Arduino - 0015

led_blinkea $

//Aquí van los comentarios generales, como por ejemplo que este código hará
//blinkear un LED por medio segundo.
//siempre deberá ir entre //, porque o si no, se hará parte del código

int redPin = 12;           // LED rojo conectado a pin digital 12
int greenPin = 11;        //LED verde conectado a pin digital 11

void setup()              // corre una vez, cuando comience sketch
{
  pinMode(redPin, OUTPUT); // setea o determina el pin digital como output
  pinMode(greenPin, OUTPUT); // setea o determina el pin digital como output
}

void loop()               // corre una y otra vez
{
  digitalWrite(redPin, HIGH); // setea o determina LED rojo encendido u on
  digitalWrite(greenPin, HIGH); // setea o determinan LED verde encendido u on
  delay(500);                // espera por medio segundo
  digitalWrite(redPin, LOW);  // setea o determina LED rojo apagado u off
  digitalWrite(greenPin, LOW); // setea o determinan LED verde apagado u off
  delay(500);                // espera por medio segundo
}
```

Done Saving.

25

// comentarios //

Declarar variables a utilizar en nuestro código

Configuración inicial del programa

Bucle o Loop
Conjunto de instrucciones
Que se repiten constantemente

Sintaxis básica

Comentarios:

```
// Esto significa que puedo comentar por  
// ejemplo, que significa esta línea en mi  
// código.
```

```
Arduino - 0015

led_blinkea $

//aca van los comentarios generales, como por ejemplo que este código hará
//blinkear un LED por medio segundo.
//siempre deberá ir entre //, porque o si no, se hará parte del código

int redPin = 12;           // LED rojo conectado a pin digital 12
int greenPin = 11;        // LED verde conectado a pin digital 11

void setup()              // corre una vez, cuando comience sketch
{
  pinMode(redPin, OUTPUT); // setea o determina el pin digital como output
  pinMode(greenPin, OUTPUT); // setea o determina el pin digital como output
}

void loop()               // corre una y otra vez
{
  digitalWrite(redPin, HIGH); // setea o determina LED rojo encendido u on
  digitalWrite(greenPin, HIGH); // setea o determinan LED verde encendido u on
  delay(500);                // espera por medio segundo
  digitalWrite(redPin, LOW);  // setea o determina LED rojo apagado u off
  digitalWrite(greenPin, LOW); // setea o determinan LED verde apagado u off
  delay(500);                // espera por medio segundo
}
```

Done Saving.

25

→ // comentarios //

→ Declarar variables a utilizar en nuestro código VARIABLES

→ Configuración inicial del Programa **FUNCIONES**

→ Bucle o LOOP
Conjunto de instrucciones
Que se repiten constantemente

Definición de Funciones

- Las funciones te permiten crear piezas modulares de código, de forma que se puedan realizar ciertas rutinas y retornar al área de código desde donde se realizó la llamada a la función.

De forma que una función se define por el tipo de valor que devuelve (enlace a declaración de variables), por su nombre, por la lista de argumentos o parámetros que le son pasados (expresión entre paréntesis) y el bloque de código que se ejecuta cuando se realiza la llamada.

El tipo "void" del ejemplo, significa nada e indica que la función no va a devolver ningún valor.

- Arduino que existan **2 funciones** llamadas:

setup()

y

loop ()

setup() es donde colocas todo el código a ejecutar una vez y al principio de tu programa.

loop() contiene el corazón de tu programa, que es ejecutado una y otra vez. Esto ocurre así porque Arduino no es como tu computador regular. No puede correr múltiples programas al mismo tiempo y estos no pueden finalizar. Cuando enchufas la placa, el código corre, cuando quieres parar, tienes que apagar todo.

Void setup()

Esta línea le da el nombre a un bloque de código. Si fueras a escribir una lista de instrucciones para enseñar a Arduino a que te pase el lápiz, escribirías *void passElLápiz()* al principio del bloque, y este bloque se convertiría en la instrucción que puedes llamar desde cualquier lugar en el código.

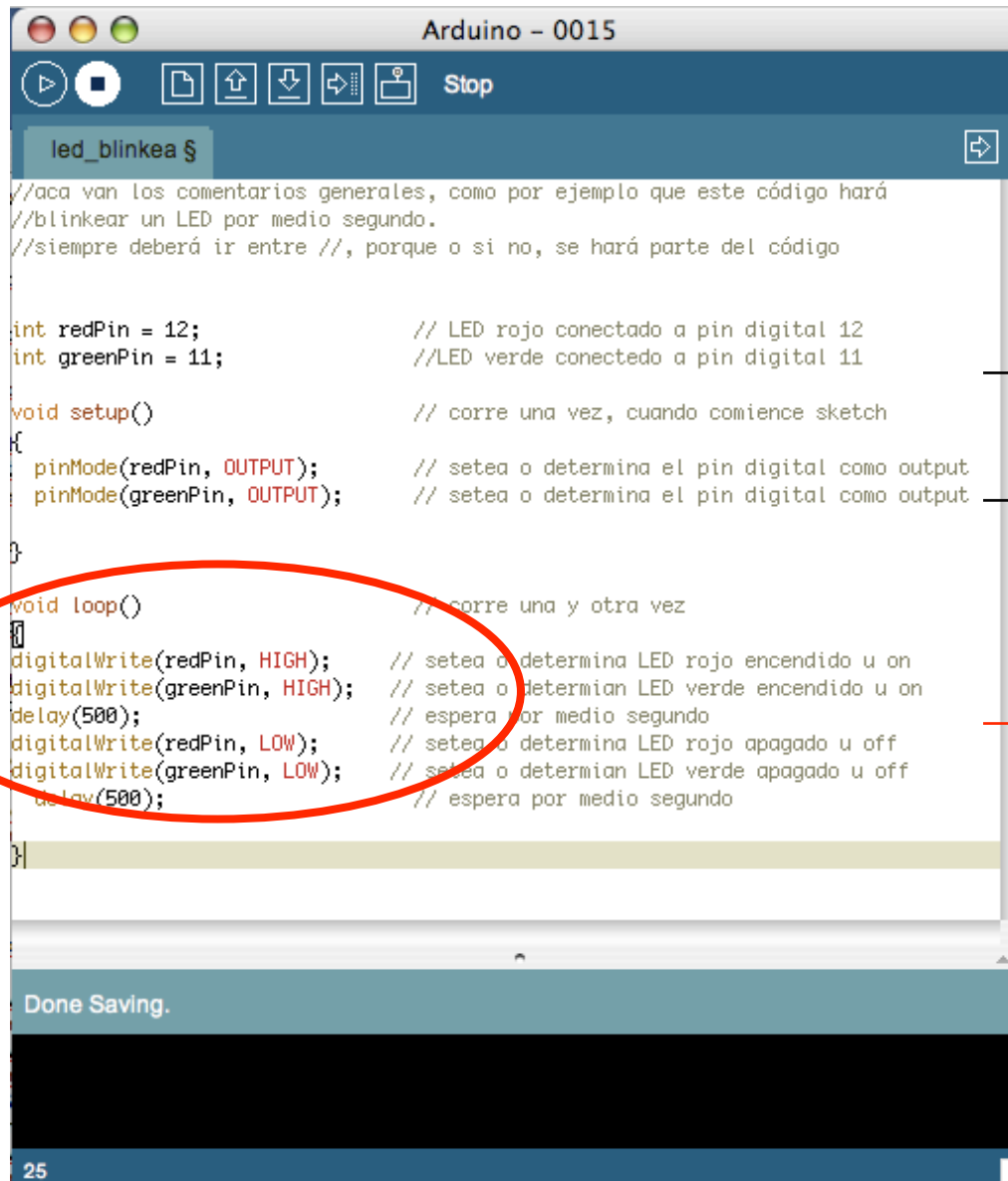
Estos bloques son llamados **funciones**. Si después de esto escribes *passElLápiz()* en cualquier lugar en tu código, Arduino ejecutara esta instrucción y continuara donde había quedado.

Void setup()

Esta línea le dice a Arduino que el próximo bloque de código se llamara *setup()*.

{ Con este paréntesis que abre, un bloque de código comienza.

} Con este paréntesis que cierra, un bloque de código termina de la función *setup()*.



```
//aca van los comentarios generales, como por ejemplo que este código hará
//blinkear un LED por medio segundo.
//siempre deberá ir entre //, porque o si no, se hará parte del código

int redPin = 12;           // LED rojo conectado a pin digital 12
int greenPin = 11;        //LED verde conectado a pin digital 11

void setup()               // corre una vez, cuando comience sketch
{
  pinMode(redPin, OUTPUT); // setea o determina el pin digital como output
  pinMode(greenPin, OUTPUT); // setea o determina el pin digital como output
}

void loop()                // corre una y otra vez
{
  digitalWrite(redPin, HIGH); // setea o determina LED rojo encendido u on
  digitalWrite(greenPin, HIGH); // setea o determin LED verde encendido u on
  delay(500);                // espera por medio segundo
  digitalWrite(redPin, LOW); // setea o determina LED rojo apagado u off
  digitalWrite(greenPin, LOW); // setea o determin LED verde apagado u off
  delay(500);                // espera por medio segundo
}
```

Declarar variables a utilizar en nuestro código VARIABLES

Configuración inicial del Programa FUNCIONES

Bucle o **LOOP**
Conjunto de instrucciones
Que se repiten constantemente

```
void loop() {
```

loop() es donde se especifica el comportamiento principal de tu device interactivo. Se repetirá una y otra vez hasta que apagues la placa.

```
Arduino - 0015

led_blinkea §

//aca van los comentarios generales, como por ejemplo que este código hará
//blinpear un LED por medio segundo.
//siempre deberá ir entre //, porque o si no, se hará parte del código

int redPin = 12;           // LED rojo conectado a pin digital 12
int greenPin = 11;        // LED verde conectado a pin digital 11

void setup()              // corre una vez, cuando comience sketch
{
  pinMode(redPin, OUTPUT); // setea o determina el pin digital como output
  pinMode(greenPin, OUTPUT); // setea o determina el pin digital como output
}

void loop()               // corre una y otra vez
{
  digitalWrite(redPin, HIGH); // setea o determina LED rojo encendido u on
  digitalWrite(greenPin, HIGH); // setea o determina LED verde encendido u on
  delay(500);                // espera por medio segundo
  digitalWrite(redPin, LOW);  // setea o determina LED rojo apagado u off
  digitalWrite(greenPin, LOW); // setea o determina LED verde apagado u off
  delay(500);                // espera por medio segundo
}
```

Done Saving.

25

→ // comentarios //

→ Declarar variables a utilizar en nuestro código **VARIABLES**

→ Configuración inicial del Programa **FUNCIONES**

→ Bucle o Loop
Conjunto de instrucciones
Que se repiten constantemente

Variables

Def.

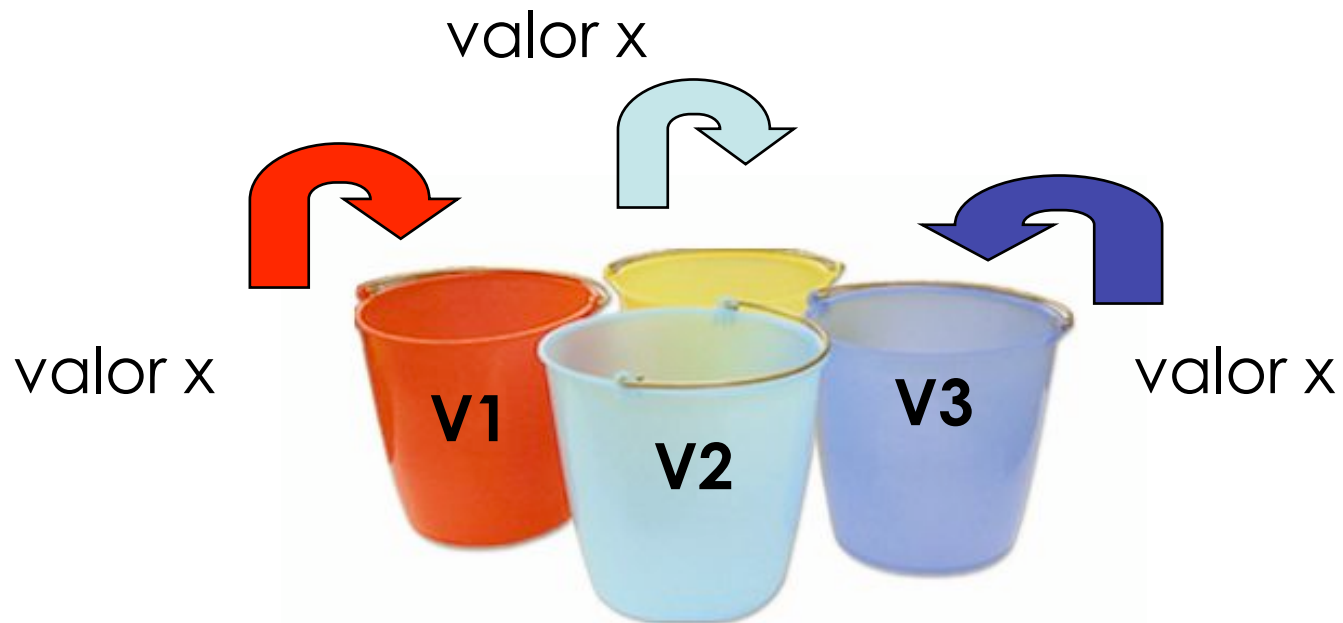
Son expresiones que **almacenan valores**, como las capturas o lecturas de los valores de entrada (input) o salida (output) de un pin analógico o sensor.



Variables

Dicho de otra forma, a **cada variable** le corresponde un **área de memoria** donde se puede **almacenar un valor**.

La unidad básica de memoria es el Byte.



Variables

Ejemplo:

```
int inputVariable = 0;
```

Declara la variable; necesario una vez.

```
inputVariable = analogRead(2);
```

le asigna a la variable el valor de entrada del pin analógico #2

Declaro una variable "**inputVariable**", y entonces se inicializa con el valor capturado desde el pin analógico numero #2.

Variables

"inputVariable" es la variable.

La primera línea es para indicar que contendrá un dato de tipo "int" (entero).

La segunda línea le asigna a la variable el valor capturado del pin analógico numero #2. Lo que hace que se pueda acceder al valor del pin #2 desde cualquier parte del código.

```
int inputVariable = 0;
```

```
inputVariable = analogRead(2);
```


Variables

- Una vez que a la variable le ha sido asignado un valor (o reasignado), se puede comprobar su valor para saber si cumple ciertas condiciones, o se puede usar su valor directamente. Por ejemplo, el siguiente código comprueba si la variable "inputVariable" es menor que 100, y realizara una pausa basada en el valor de dicha variable, la cual tendrá siempre como mínimo un valor de 100ms:
- `if (inputVariable < 100) { inputVariable = 100 }`
- `delay(inputVariable)`

Variables

- `if (inputVariable < 100) { inputVariable = 100 }`
- `delay(inputVariable)`

Este ejemplo muestra las tres operaciones que se pueden realizar con variables.

1. Comprueba el valor de la variable
(`if (inputVariable < 100)`)
2. Le asigna un valor a la variable si cumple la condición
(`inputVariable = 100`)
3. Usa el valor de la variable como entrada para la función `delay()`
(`delay(inputVariable)`)

Variables

- Deberás dar a tus variables nombres descriptivos, de forma que hagas tu código más inteligible. Los nombres de las Variables como "tiltSensor" (sensor tilt) o "pushButton" (pulsador), te pueden ayudar (y a cualquiera que lea tu código) a entender lo que la variable representa. Los nombres de las Variables como "var" o "value" (valor) , por ejemplo, hacen el código un poco más inteligible.

Variables

- Las variables tienen que ser declaradas siempre antes de ser usadas. Declarar una variable significa que hay que definir su tipo, y darle un valor inicial. En el ejemplo de arriba, la declaración

int inputVariable = 0;

declara que la variable "inputVariable" es de tipo "int" (entero corto), y que su valor inicial es cero.

Variables

Posibles tipos para las variables son:

- **int** : Usan 2 bytes de memoria para representar un numero entre -32,768 y 32,767; es el tipo de dato mas común utilizado en Arduino.
- **char** : puede contener un solo caracter, como A. Como cualquier computador, Arduino lo guarda como un numero, aun cuando tu ves texto, los valores van de -128 a 127.
- **boolean**: puede tener 2 valores, Verdadero o Falso.
-
- **array** : Es una lista de variables a las que se puede entrar por un indice, se usan para crear tablas de valores a las que puedas acceder facilmente. Por ejemplo, si quieres guardar diferentes valores de brillo para un LED, podrias crear seis variables llamadas light01, light02, etc. Mejor aun, podrias usar un array como este `int light[6] = {0, 20, 50, 75, 100};` la palabra "array" no es usada en la declaracion de variables porque es el simbolo `[]` y `{}` los que hacen el trabajo.
- **byte** : Contiene un nuemro de 0 a 255. Como los "chars", los bytes usan solo un byte de memoria.

...0

* int – Un entero. Números enteros solamente. int código postal= 11710;

* boolean – Verdadero (1) o Falso (0). boolean algoBooleano = verdadero;

* float – Un número con un punto decimal. 1.5, 3.987 etc. Temperatura es float= 27.5*;

* byte – Un valor de 8 bits. El rango es de -127 a 128 en valor. byte miByte = -90;

* char – Un sólo carácter. char mi inicial = 'C';

Operadores Relacionales o de Comparación:

Cuando especificas condiciones o pruebas para *if*, *while*, y *for* estos son los operadores que puedes usar :

- == igual a
- != no igual a
- < menor que
- > mayor que
- <= menor que o igual a
- >= mayor que o igual a

Operadores Lógicos o Booleanos:

Usados cuando quieras combinar múltiples condiciones, por ejemplo, si quieres chequear si el valor que viene de un sensor esta entre 5 y 10, escribirías:

```
if ((sensor ==> 5) && (sensor <= 10))
```

Hay tres operadores: **AND**, representado por **&&**; **OR**, representado por **| |**; y **NOT** representado por **!**

2.6. Estructuras de programación

- **while** Similar a *if*, ejecuta un bloque de código, mientras cierta condición sea cierta. Por ejemplo:

```
// LED blinkea mientras sensor esta bajo 512
```

```
sensorValue = analogRead(1);  
while (sensorValue < 512) {  
  digitalWrite(13,HIGH);  
  delay(100);  
  digitalWrite(13,HIGH);  
  delay(100);  
  sensorValue = analogRead(1); }
```


Estructuras de programación

do ... while Asi como *while*, excepto que el código corre justo antes que la condición sea evaluada. Se usa cuando quieres hacer código dentro de tu bloque para hacerlo correr al menos una vez antes de chequear las condiciones.

Ejemplo:

```
do {  
  digitalWrite(13,HIGH);  
  delay(100);  
  digitalWrite(13,HIGH);  
  delay(100);  
  sensorValue = analogRead(1);  
} while (sensorValue < 512);
```

Estructuras de programación

if ... else Esta estructura toma decisiones en tu programa, debe ir seguido por una pregunta específica como una expresión contenida entre paréntesis. Si la expresión es Verdadera, lo que sea que siga, será ejecutado. Si es falsa, el bloque de código siguiente a *else* será ejecutado. Es posible usar solo *if* sin la cláusula *else*.

Por ejemplo:

```
if (val == 1) { digitalWrite(LED,HIGH); }
```

for te permitirá repetir un bloque de código un número específico de veces.

Por ejemplo:

```
for (int i = 0; i < 10; i++) { Serial.print("hola"); }
```

**Funciones básicas de manejo de Arduino:
Definición de I/O, valores digitales de I/O**

```
//aca van los comentarios generales, como por ejemplo que este código hará
//blinquear un LED por medio segundo.
//siempre deberá ir entre //, porque o si no, se hará parte del código

int redPin = 12;           // LED rojo conectado a pin digital 12
int greenPin = 11;        //LED verde conectado a pin digital 11

void setup()               // corre una vez, cuando comience sketch
{
  pinMode(redPin, OUTPUT); // setea o determina el pin digital como output
  pinMode(greenPin, OUTPUT); // setea o determina el pin digital como output
}

void loop()                // corre una y otra vez
{
  digitalWrite(redPin, HIGH); // setea o determina LED rojo encendido u on
  digitalWrite(greenPin, HIGH); // setea o determinan LED verde encendido u on
  delay(500);                // espera por medio segundo
  digitalWrite(redPin, LOW);  // setea o determina LED rojo apagado u off
  digitalWrite(greenPin, LOW); // setea o determinan LED verde apagado u off
  delay(500);                // espera por medio segundo
}
```

Done Saving.

```
pinMode(LED, OUTPUT); // setea el pin digital como output
```

pinMode le dice a Arduino como configurar un cierto pin.

Los pines digitales pueden ser usados como **INPUT** o **OUTPUT**.

En este caso, necesitamos un pin como output para controlar el LED, para que podamos poner el número del pin y su modo dentro del paréntesis.

pinMode es una función, y las palabras (o números) especificados dentro del paréntesis son **argumentos**. INPUT y OUTPUT son constantes en el lenguaje de Arduino.

```
//aca van los comentarios generales, como por ejemplo que este código hará
//blinkear un LED por medio segundo.
//siempre deberá ir entre //, porque o si no, se hará parte del código

int redPin = 12;           // LED rojo conectado a pin digital 12
int greenPin = 11;         //LED verde conectado a pin digital 11

void setup()               // corre una vez, cuando comience sketch
{
  pinMode(redPin, OUTPUT); // setea o determina el pin digital como output
  pinMode(greenPin, OUTPUT); // setea o determina el pin digital como output
}

void loop()                // corre una y otra vez
{
  digitalWrite(redPin, HIGH); // setea o determina LED rojo encendido u on
  digitalWrite(greenPin, HIGH); // setea o determinan LED verde encendido u on
  delay(500);                // espera por medio segundo
  digitalWrite(redPin, LOW);  // setea o determina LED rojo apagado u off
  digitalWrite(greenPin, LOW); // setea o determinan LED verde apagado u off
  delay(500);                // espera por medio segundo
}
```

Done Saving.

```
digitalWrite(LED, HIGH); // enciende el LED
```

digitalWrite() hace posible encender (y apagar) cualquier pin que ha sido configurado como OUTPUT. El primer argumento (en este caso *LED*) especifica cual pin debería estar on u off (recuerda que *LED* es una valor constante que se refiere al pin 13). El segundo argumento puede poner el pin on (HIGH) u off (LOW). Imagina que cada pin output es un pequeño soquete de 220 V, y Arduino trabaja a modestos 5 V. La magia aquí es cuando el software se transforma en hardware. Cuando escribes *digitalWrite(LED, HIGH)*, coloca el output pin en 5 V, y si conectas un LED, este se encenderá. En este punto en tu código, una instrucción en el software hace que algo pase en el mundo físico por medio del control del flujo de la electricidad al pin. Poniendo el pin on y off nos permitirá traducir esto en algo mas visible para un ser humano: el LED es nuestro actuador.

```
//aca van los comentarios generales, como por ejemplo que este código hará
//blinkear un LED por medio segundo.
//siempre deberá ir entre //, porque o si no, se hará parte del código

int redPin = 12;           // LED rojo conectado a pin digital 12
int greenPin = 11;        //LED verde conectado a pin digital 11

void setup()              // corre una vez, cuando comience sketch
{
  pinMode(redPin, OUTPUT); // setea o determina el pin digital como output
  pinMode(greenPin, OUTPUT); // setea o determina el pin digital como output
}

void loop()               // corre una y otra vez
{
  digitalWrite(redPin, HIGH); // setea o determina LED rojo encendido u on
  digitalWrite(greenPin, HIGH); // setea o determinan LED verde encendido u on
  delay(500);                // espera por medio segundo
  digitalWrite(redPin, LOW);  // setea o determina LED rojo apagado u off
  digitalWrite(greenPin, LOW); // setea o determinan LED verde apagado u off
  delay(500);                // espera por medio segundo
}
```

Done Saving.


```
delay(1000);    // espera por un segundo
```

Arduino tiene una estructura muy básica. Si quieres que las cosas pasen con cierta regularidad, le dices paso a paso que hacer.

delay() básicamente le dice al procesador que espere y no haga nada por una cantidad de milisegundos que pasan como un argumento. Milisegundos son miles de segundos. 1000 milisegundos = 1 segundo. Así que el LED se queda on por un segundo.

```
//aca van los comentarios generales, como por ejemplo que este código hará
//blinkear un LED por medio segundo.
//siempre deberá ir entre //, porque o si no, se hará parte del código

int redPin = 12;           // LED rojo conectado a pin digital 12
int greenPin = 11;         //LED verde conectado a pin digital 11

void setup()               // corre una vez, cuando comience sketch
{
  pinMode(redPin, OUTPUT); // setea o determina el pin digital como output
  pinMode(greenPin, OUTPUT); // setea o determina el pin digital como output
}

void loop()                // corre una y otra vez
{
  digitalWrite(redPin, HIGH); // setea o determina LED rojo encendido u on
  digitalWrite(greenPin, HIGH); // setea o determinan LED verde encendido u on
  delay(500);                // espera por medio segundo
  digitalWrite(redPin, LOW);  // setea o determina LED rojo apagado u off
  digitalWrite(greenPin, LOW); // setea o determinan LED verde apagado u off
  delay(500);                // espera por medio segundo
}
```

Done Saving.

```
digitalWrite(LED, LOW);    // pone LED off
```

Esta instrucción pone el LED en off que previamente pusimos en on.

Por que usamos HIGH y LOW? Porque es una antigua convencion el electronica digital.

HIGH significa que el pin esta on, y en el caso de Arduino, se seteara en 5 V.

LOW significa que el pin esta off, seteado en 0 V.

```
//aca van los comentarios generales, como por ejemplo que este código hará
//blinkear un LED por medio segundo.
//siempre deberá ir entre //, porque o si no, se hará parte del código

int redPin = 12;           // LED rojo conectado a pin digital 12
int greenPin = 11;         //LED verde conectado a pin digital 11

void setup()               // corre una vez, cuando comience sketch
{
  pinMode(redPin, OUTPUT); // setea o determina el pin digital como output
  pinMode(greenPin, OUTPUT); // setea o determina el pin digital como output
}

void loop()                // corre una y otra vez
{
  digitalWrite(redPin, HIGH); // setea o determina LED rojo encendido u on
  digitalWrite(greenPin, HIGH); // setea o determinan LED verde encendido u on
  delay(500);                // espera por medio segundo
  digitalWrite(redPin, LOW); // setea o determina LED rojo apagado u off
  digitalWrite(greenPin, LOW); // setea o determinan LED verde apagado u off
  delay(500);                // espera por medio segundo
}
```

Done Saving.

Para resumir, este programa hace esto:

- Hace que el pin 13 sea un output
(solo una vez al principio)
- Entra un loop
- Coloca el LED conectado al pin 12 en on.
- Espera por un segundo.
- Coloca el LED conectado al pin 11 en off.
- Espera por un segundo.
- Va hacia atrás, al principio del loop.